A Parallel, Adaptive Refinement Scheme for Tetrahedral and Triangular Grids

Alan Stagg[1], Jackie Hallberg[2], and Joseph Schmidt[3]

[1] Los Alamos National Laboratory, Applied Physics Division
P.O. Box 1663, MS P365
Los Alamos, NM 87545
stagg@lanl.gov
[2] U.S. Army Engineer Research and Development Center
Coastal and Hydraulics Laboratory
3909 Halls Ferry Road
Vicksburg, MS 39180
pettway@juanita.wes.army.mil
[3] 2420 Wanda Way
Reston, VA 20191
roig.and.schmidt@erols.com

Abstract. A grid refinement scheme has been developed for tetrahedral and triangular grid-based calculations in message-passing environments. The element adaption scheme is based on edge bisection of elements marked for refinement by an appropriate error indicator. Hash table/linked list data structures are used to store nodal and element information. The grid along inter-processor boundaries is refined consistently with the update of these data structures via MPI calls. The parallel adaption scheme has been applied to the solution of a transient, three-dimensional, nonlinear, groundwater flow problem. Timings indicate efficiency of the grid refinement process relative to the flow solver calculations.

1 Introduction

Adaptive grid methods based on point insertion and removal have been popular for a number of years for achieving greater solution accuracy with relative cost efficiency. However, issues related to implementing such schemes on parallel systems are just now being addressed, and much work is needed to identify the best approaches.

In this paper we present a new approach for the h-refinement of irregular tetrahedral and triangular grids in message-passing environments. Data structures have been selected to simplify implementation and coding complexity as much as possible for refinement, coarsening, and load balancing components. This software is being implemented in the Department of Defense code ADH (ADaptive Hydrology) under development at the U.S. Army Engineer Research

and Development Center. ADH is a modular, parallel, finite element code designed to support groundwater, surface water, and free-surface Navier-Stokes modeling [1].

1.1 Serial Element Adaption Scheme

Given an initial grid, the model subdivides grid elements to achieve the desired resolution in regions of interest. The parallel grid adaption scheme developed here is based on the geometric splitting algorithm of Liu and Joe [2]. Elements are refined by edge bisection according to an error indicator, and elements can be merged to increase efficiency where high resolution is not required. A grid closure step is utilized to eliminate *hanging* nodes. Element edges are selected for bisection based on a modified longest-edge bisection approach in which the oldest edge in the element is first flagged for bisection followed by the longest edge. Refinement and coarsening of a tetrahedral element are illustrated in Figure 1. Here a new node is added to an edge, creating two new tetrahedra. The new elements can be merged to recover the original element by removing the inserted node.

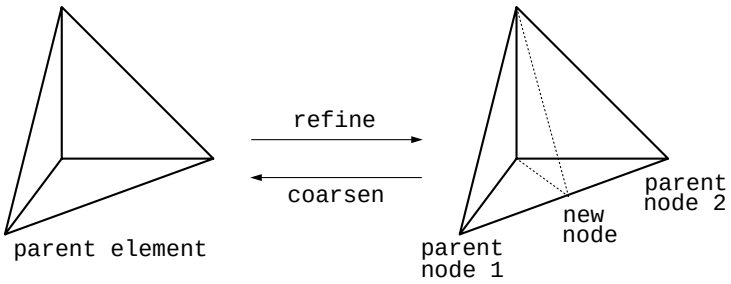

Fig. 1. Tetrahedral Grid Adaption Based on Edge Bisection

The refinement process begins with determination of elements to be split according to an explicit error indicator. The following pseudo-code illustrates the basic steps in the refinement scheme.

Refinement pseudo-code

```
loop over elements
    refine element via edge bisection if its error > tolerance;

conforming_grid = false;
do while conforming_grid == false {
    conforming_grid = true;
```

```
    loop over elements
         if element has an edge with a newly inserted node {
              refine element;
              conforming_grid = false;
         }
}
```

2 Parallel Implementation

The parallel implementation of local grid refinement schemes like the edge bisection scheme presented above presents a number of challenges for the message-passing environment. First, in the standard approach where the grid is partitioned and subregions are assigned to processors, the subregions must be refined and coarsened consistently along processor boundaries. Also, closure requirements may force element refinement to spread to a processor that has no elements marked for refinement by the error indicator. Finally, the local adaption process will likely lead to load imbalances among the processors, and nodes and elements must be transferred between processors during dynamic load balancing so that processing efficiency is maintained. In this paper we focus on describing the methodology developed for element refinement in parallel.

In our approach the grid is partitioned by assigning element nodes to processors. Elements along processor boundaries are shared by the processors owning the element nodes. Nodal information for these elements is communicated between processors using MPI, and each processor stores complete data for its shared elements [3].

2.1 Data Structures

Data structures were selected to simplify the parallel implementation of the adaption scheme and to facilitate the coupling of the refinement, coarsening, and load balancing components. During early work, we realized that common techniques like the use of tree structures for refinement could adversely impact other adaption components such as load balancing. In this case, the use of graph partitioners and the resulting grid point movement between processors requires splitting refinement trees between processors. To avoid these difficulties we use hash table/linked list structures [4]. Such structures are naturally suited for grid adaption since they are dynamic in nature and facilitate node and element searches. These structures handle all grid refinement, coarsening, and load balancing needs without complicating the implementation of any single component.

Hash tables are used to store nodes and element edges. Each entry in the node hash table consists of a local node number relative to the owning processor and corresponding node identifier in the global grid. Each entry in the edge hash table consists of the two local node numbers that define the edge, an integer edge rank based on comparative lengths of the edges, and an integer that stores

the new node number if a node is inserted on the edge. Prior to refinement, the node and edge hash tables are allocated and filled, and the memory is freed once the refinement process is complete.

2.2 Grid Consistency between Processors

The grid refinement scheme presented here is primarily a local process and is thus amenable to parallel processing. The principal requirement in the parallel environment is that processors periodically communicate to maintain grid consistency along the inter-processor boundaries. One example is illustrated in Fig. 2 for the communication of edges. Here two elements are shown with nodes

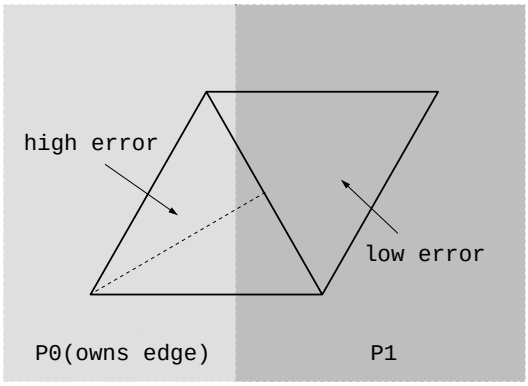

Fig. 2. Edge Bisected by Owning Processor

distributed over two processors. The left element is marked for refinement due to high error while the right element is not. Processor 0 bisects the common edge, and processor 1 must be informed that refinement of the right triangle is required for closure.

In the serial case, the new node number on the edge is stored in the edge hash table, and the adjacent element checks for the presence of a new node in the hash table to see if refinement for closure is required. In our partitioning approach, an edge that spans two processors will appear in each of these processors' hash tables, and a protocol must be established to maintain consistency of the edge hash tables between processors. To support this communication, edge communication lists are constructed which provide a mapping between these duplicated edge storage locations. For each such edge, one of the processors sharing the edge is assigned ownership of it.

2.3 Edge Ranking

After constructing the edge communication lists, the edges are ranked based on length so that they are uniquely and consistently identified throughout the

global grid for the refinement phase. Integer rankings are utilized rather than using computed edge lengths so that processors are easily able to make consistent edge bisection decisions when multiple edges in an element are the same length.

Following a parallel odd-even transposition sort, global ranks are returned to processors owning the edges, and the ranks are then stored by these processors in their edge hash tables. These processors then communicate the ranks to the processors sharing the edges using the edge communication lists that have been constructed. The receiving processors finally store the ranks in their edge hash tables.

2.4 Element Refinement

After elements have been selected for refinement based on the error indicator, edges are selected for bisection based on their age and ranking within the element. To refine an element, the oldest edge (or longest edge in a tie) is bisected if necessary. To determine if the edge has already been bisected, the new node entry in the edge structure is inspected for that edge. If the edge has not been bisected, a new node is created for the edge, and the hash table is adjusted locally. Two new elements are created with the bisection of an edge, and the element Jacobians and other data are corrected for these elements. The new node entries for the edges in these new elements are reinitialized to indicate that new nodes are not present.

2.5 Grid Closure

After elements have been refined based on the error indicator, further refinement might be required to obtain a conforming grid. In the serial case each element is checked for edges with new nodes via the edge hash table. If any element has an edge with a new node, that element is marked for refinement according to the established rules. The refinement process continues iteratively until a closed grid is obtained.

In the parallel environment this procedure is complicated by the fact that shared edges may be bisected by only one of the processors spanned by the edge. To maintain consistency of the edge hash tables, processors owning shared edges communicate new node information to processors sharing the edges. If a message indicates that an edge has a new node, then the receiving processor creates a new node for the edge and updates its hash table. Similarly, processors may bisect edges they do not own. To handle this situation, the edge communication lists are utilized in reverse order (the send list becomes a receive list, and vice verse) so that processors owning shared edges can update their hash tables if other processors bisect them. After this communication, the elements with new nodes on edges are refined, and the process is repeated until the grid is closed.

3 Results

The capabilities of the parallel grid refinement scheme have been investigated for the solution of a draining heterogeneous column. In this problem a column is filled with a mixture of clay, silt, and sand. The column consist primarily of sand with a clay lens near the bottom and silt lenses in several places throughout the column. Initially, the column is completely saturated with water, and then the water is allowed to drain from the bottom of the column. The grid is allowed to refine and coarsen locally as dictated by the explicit error indicator, and dynamic load balancing is utilized to improve processor efficiency.

A snapshot of the adaptively refined grid for the hegerogeneous column is illustrated in Fig. 3. The area shaded black represents the clay material, while

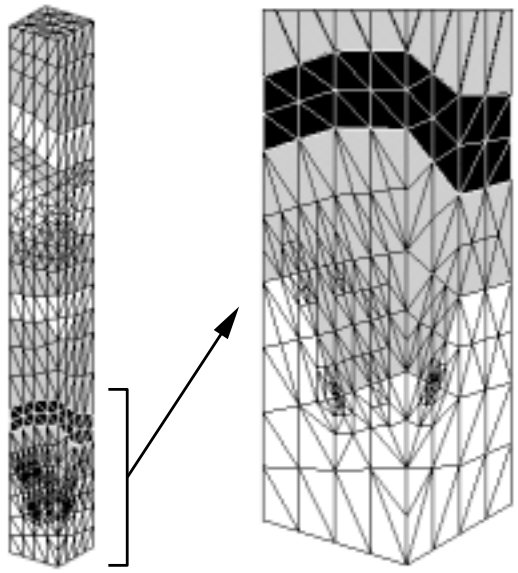

Fig. 3. Adaptively Refined Grid for Heterogeneous Column

the sand and silt are represented by the gray and white regions, respectively. Grid refinement is visible at the sand/silt interface in the lowest points of the sand. This refinement is indicative of the large head gradient from the sand to the silt. Water travels through the sand at a faster rate than through the silt due to the silt's lower conductivity. As a result, the water ponds, or collects, in the low points of the sand until the pressure is great enough to push the flow across the interface.

4 Conclusion

A parallel refinement scheme has been developed for tetrahedral and triangular grids. The refinement scheme and data structures described in this abstract have been developed to facilitate the parallel implementation of both grid refinement and coarsening. Though not described here, the coarsening phase (like the refinement phase) is based on communicating a minimum set of data and reconstructing information locally where necessary without the use of tree structures. The goal with this approach is a balanced design between refinement, coarsening, and load balancing in terms of efficiency and ease of implementation. Application of the adaptive grid scheme to a transient groundwater flow problem has demonstrated the capabilities and efficiency of the method.

References

1. Jenkins, E.W., Berger, R.C., Hallberg, J.P., Howington, S.E., Kelley, C.T., Schmidt, J.H., Stagg, A.K., and Tocci, M.D., "Newton-Krylov-Schwarz Methods for Richards' Equation," submitted to the *SIAM Journal on Scientific Computing*, October 1999.
2. Liu, A. and Joe, B., "Quality Local Refinement of Tetrahedral Meshes Based on Bisection," *SIAM Journal on Scientific Computing*, vol. 16, no. 6, pp. 1269-1291, November 1995.
3. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., and Dongarra, J., *MPI- The Complete Reference, Volume 1, The MPI Core*, The MIT PRess, Cambridge, Massachusetts, 1998.
4. Cormen, T., Leiserson, C., Rivest, R., *Introduction to Algorithms*, The MIT Press, Cambridge, Massachusetts, 1990.